SAND2016-7042C

Spiking Network Algorithms for Scientific Computing

William Severa, Ojas Parekh, Kristofor D. Carlson, Conrad D. James, James B. Aimone
Center for Computing Research, Sandia National Laboratories
Albuquerque, NM, USA

***Abstract*—For decades, neural networks have shown promise for next-generation computing, and recent breakthroughs in machine learning techniques, such as deep neural networks, have provided state-of-the-art solutions for inference problems. However, these networks require thousands of training processes and are poorly suited for the precise computations required in scientific or similar arenas. The emergence of dedicated spiking neuromorphic hardware creates a powerful computational paradigm which can be leveraged towards these exact scientific or otherwise objective computing tasks. We forego any learning process and instead construct the network graph by hand. In turn, the networks produce guaranteed success often with easily computable complexity. We demonstrate a number of algorithms exemplifying concepts central to spiking networks including spike timing and synaptic delay. We also discuss the application of cross-correlation particle image velocimetry and provide two spiking algorithms; one uses time-division multiplexing, and the other runs in constant time.**

I. Introduction

Neural computing technologies have been one of several proposed novel architectures to either replace or complement the ubiquitous von Neumann architecture platform that has dominated conventional computing for the last 70 years [1], [2], [3], [4]. Despite its biological substrate, the brain's ability to perform sophisticated, non-trivial computation at very low energy costs has intrigued computer scientists back to Alan Turing and John von Neumann [5], [6]. Unlike quantum computing, which is also mentioned as a post-Moore's Law alternative architecture, neural computation does not rely on novel physics with which to perform the computation, but rather relies on a combination of factors that make it distinct from more conventional processing approaches. While there is continual discussion as to what aspects of biological neural systems should be used to inspire computing, the most common sources of inspiration are spike-based computation, colocalization of processing and memory at synapses, the brain's large scale and extensive connectivity, and the brain's robust ability to learn.

Not surprisingly, it is this learning ability of biological neural systems that has most often captured the imagination of advocates of neural computation. Indeed, the increased emphasis on data-centric computations such as image processing and audio recognition has led to renewed interest in machine learning methods such as deep artificial neural networks (ANNs) which are loosely related to information processing in the sensory cortex of the brain [7], [8], [9], [10]. A further source of excitement is that these methods have achieved dramatic success despite only beginning to incorporate aspects of the brain's considerable complexity. For instance the brain's ability to learn relies on numerous mechanisms, ranging from synaptic plasticity to synapto- and neurogenesis [11], [12], which despite becoming increasingly well understood biologically have been minimally explored in the computational domain.

Despite this dominant focus on learning algorithms, there are several reasons that more conventional numerical algorithms may be worth considering as an application for neural computing. First, hardware-level learning continues to be a technical challenge, whereas other aspects of neural computing, such as spiking, are more readily implementable in current CMOS technology [13]. Second, while learning algorithms are approximate in nature, and thus often considered inappropriate for scientific computation; there are exact models of neural computation that are more suitable, such as threshold gate circuits, which are very similar to spiking neural circuits [14], [15]. Finally, scientific computation remains an important driver for computing technologies, and impactful advances on neurally accelerating key numerical kernels could presumably have significant impact in more data-centric applications such as graph analytics and dimensionality reduction.

Here, we describe three simple spike-based neural algorithms that utilize connectivity in an explicitly "hand-crafted" manner to perform reasonably complex computational kernels. The intent of these vignettes is to illustrate how relatively straightforward configurations of spiking neurons can allow the very rapid, parallel computation of common functions that are not necessarily well fit to the serial von Neumann systems. Furthermore, these algorithms are structured so as to be readily implementable in neural hardware being discussed today.

II. Preliminaries

A. Spiking Networks

Spiking neural networks have a long history of being used in neuroscience to simulate biological systems [21], [22], [23]. A spiking neural network differentiates itself from other neural networks largely by its method of neuron-to-neuron

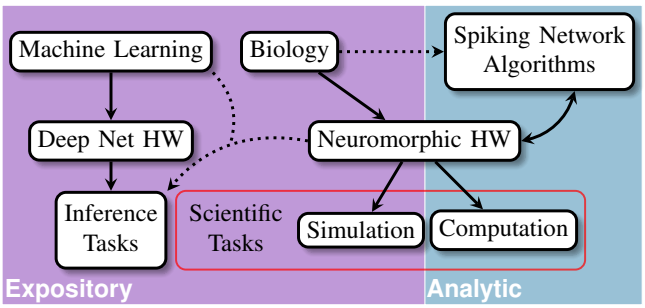

Fig. 1. Layout of machine learning and neural computing. Spiking networks algorithms sit distinct from the trained networks of machine learning and biologically-inspired simulations. However, spiking network algorithms can leverage neuromorphic hardware originally designed for these other applications.

TABLE I
A COLLECTION OF NEUROMORPHIC HARDWARE SPECIFICATIONS

	Process (nm)	Die Size (mm^2)	Neurons	Energy (J/SynEvent)
SpiNNaker [16], [17]	130	102	18k	8n
TrueNorth [4]	28	430	1M	26p
HICANN [18]	180	50	512	-
NeuroGrid [19]	180	168	65k	941p
IFAT [20]	90	16	65k	22p

communication. Incorporating a time dimension for signaling allows for more accurate representation of the spiking behavior of biological neurons. Each biological neuron has an internal voltage which varies over time. A connection between neurons is called a synapse. Synapses are assumed to be unidirectional with a pre-synaptic and post-synaptic neuron. In the event that the voltage in the pre-synaptic neuron exceeds some predetermined threshold, the neuron sends a single-state all-or-nothing signal called a spike. This spike travels down a conduit called the axon via a biochemical pump, and is received by the post-synaptic neuron. This transmission is not instantaneous, and the time taken is represented by a synaptic delay. Post-synaptic neurons integrate incoming spikes, and this influences the voltage of the post-synaptic neuron. Depending on the synaptic weight, the influence can vary in magnitude and sign. After a spike, the voltage in the pre-synaptic neuron returns to a resting value, and the pre-synaptic neuron experiences a refractory period momentarily after a spike during which it cannot spike again. These are the basic characteristics that are simulated in a SNN, but for our purposes we abstract away from many of the details and use a basic, functional spiking neuron model as detailed below.

There are a variety of reasons biological systems utilize spike-based computation. Neural systems are structured into highly organized computational modules composed of millions of low-precision processing elements (neurons) that require massive intercommunication to perform intrinsically parallel algorithms. Due to their massive parallelism, neural systems have a balance between computation and communication that is weighted more towards communication when compared to traditional von Neumann architecture platforms. The usage of spikes dramatically reduces communication costs, thus enabling biological nervous systems to be extremely energy efficient (8-9 orders of magnitude better than digital computation [24]). Many of these factors and benefits are captured by state-of-the-art neuromorphic hardware, see Table I. Additionally, spike-based computations are a type of event-based communication protocol and can be designed to send information only when events occur. This is efficient from both an energy and information theoretic standpoint; only changes in the external world need to be communicated to the neural system for processing. A number of neuromorphic devices have already implemented event-based communications by using address-event representation (AER), a communication protocol that represents each spike with the address of the neuron that fired the spike and the time at which the spike occurred [25], [26].

In this context we adopt the structure and capabilities of a spiking neural network, but we are not held to any biologically realistic connections or parameters. Thereby we aim to extend the benefits of the tailored hardware to larger and more general applications. To reinforce the differences from both traditional artificial neural networks and biological simulation networks, we use the term *Spiking Networks*.

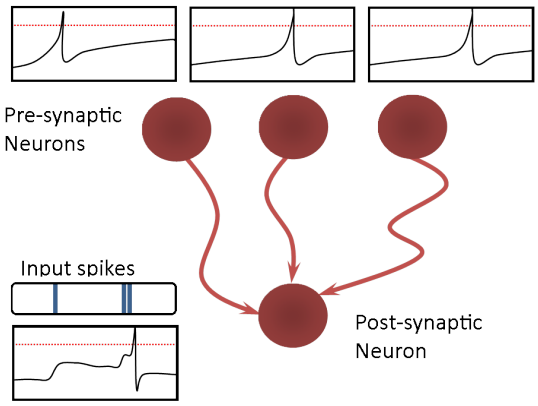

Fig. 2. An example illustration of three synapses. Pre-synaptic neurons form the upper layer and, in this case, feed into a single post-synaptic neuron. The graphs show example voltages for each neuron over time with the red lines being the spike threshold. Despite the analog behavior of the voltage within the pre-synaptic neurons, the post-synaptic neuron only receives all-or-nothing signals (the spikes) as shown in the heading 'Input spikes'.

There exist many models of individual neuron behavior. We adopt a very generic integrate-and-fire neuron model with multiplicative leakage similar to those in [27]. In biological neurons, without outside influence, the internal voltage decreases (leaks) towards a resting state. This leakage is reflected in more general neuron models. The exact nature of the decay is dependent on the specific implementation. In the majority of this paper, the type of leakage is unimportant. As such, the algorithms presented here can be implemented in software spiking neural networks or neuromorphic spiking hardware provided the basic requirements of the algorithms are satisfied.

We will note any situation where specific leakage is required, otherwise we assume no leakage.

B. Conventions

Standardized methods used to describe serialized algorithms (flow charts, pseudocode, etc.) fall short when describing spiking neural network algorithms. Traditional algorithms are serial with parallel modules, whereas a spiking algorithm is inherently parallel. Moreover, memory storage and computation occurs at the same location, not as separate operations, and these operations are performed continuously and locally at each neuron. As such, the flowcharts and pseudocode to which we have become accustomed are unfortunately misguided in this context. Instead, the connections between neurons are paramount, and so we must provide new notation for describing such connections. We will also provide an overall figure showing the connection graph and, at times, detail sample propagation of signal throughout the network.

Formally, we take a neuron to be a triple defined by (threshold, reset voltage, leakage constant) and a synapse to be a 4-tuple (pre-synaptic neuron, post-synaptic neuron, weight, delay). In describing a synapse we use the notation

$$S_{a,b} = (d, w)$$

to represent a synapse where a is the pre-synaptic neuron, b is the post-synaptic neuron, the synaptic weight is w and the delay is d. As shorthand, let $w_{i,j}$ and $d_{i,j}$ be the weight and delay respectively for the synapse $S_{i,j}$, and let T_i, R_i, m_i be the threshold, reset voltage, leakage constant respectively of neuron i. If not specified, we assume the leakage constant to be 1. A spiking network is special case of a labeled graph (N, S), where N is a set of neurons and S is a set of synapses-labels. We use neurons as indices of S.

A subset of N is called a neuron group. The term group is used only to maintain comparison with other contexts; there is no inherent algebraic structure or induced group action. For a neuron group, we adopt vector (or matrix if more convenient) notation, starting indexing at 0. Following this notation,

$$\forall i,\ S_{G_i,H_i} = (d, w)$$

signifies connecting neuron G_i to neuron H_i for all i.

There are two types of neurons whose behavior is programmatically determined. First, we have input neurons that spike at times representative of the input data. We assume input data will be provided at run-time, and that the spike times of the input neurons are directly determined by the input data. Second, we have scheduled neurons that spike at predetermined times. These can be thought of as input neurons with static input.

For all other neurons, we use a discrete-time, difference equation approach in executing the algorithm. Each neuron $y \in N$ is assigned a voltage $V_{0,y}$, and for all our algorithms we have $V_{0,y} = 0$. At time step t, neuron y computes

$$\hat{V}_y = m_y V_{t-1,y} + \sum_{S_{x,y} \text{ exists}} w_{x,y} x_{t-d_{x,y}}$$

where $x_{t-d_{x,y}} = 1$ if x spiked at time $t - d_{x,y}$ and 0 otherwise. If $\hat{V}_y \geq T_y$ neuron y spikes at time t and sets $V_{t,y}$ to the reset voltage. If $\hat{V}_y < T_y$, then $V_{t,y} = \hat{V}_y$.

C. Performance Analysis

Prior to discussing the algorithms, we quickly address performance analysis. As we develop these novel algorithms it is of course important that they be benchmarked. In other contexts, spike-based approaches running on neuromorphic hardware have shown drastic energy benefits [28]. However, appropriately benchmarking algorithms on neuromorphic hardware in general is still an open, future area of research. Honest comparisons either between neuromorphic and von Neumann architectures or between two neuromorphic platforms are fraught with hardware-dependent incompatibilities. Though to our benefit, the energy consumption of current neuromorphic hardware platforms is generally well-documented (see Table I) and we can compute the number of spikes required by our algorithms (see Table IV), thereby estimating the energy consumption. Regardless we hope to expand and provide experimental performance results in the future.

III. Spiking Network Concepts

Neuromorphic hardware is inherently different than a traditional von Neumann architecture. It is possible to construct traditional logic gates using spiking networks [29], [30], and there has been some success in adapting traditional machine learning algorithms to spiking networks [28]. However, we posit that to make the most of this new hardware, algorithms must be designed from a spiking, neural perspective at the outset. Here we explore some small, example algorithms which exemplify qualities uniquely tied to the spiking regime.

We provide two examples of increasing complexity to demonstrate a crucial difference between spiking and non-spiking systems—the ability and necessity to manage spike timing. The time dimension provides greater flexibility in algorithm design, but also requires careful synchronization. The first example is a filter which uses a neuron's leakage to gate incoming spikes. Hand-in-hand with spike timing is synaptic delay, the concept that spikes do not transmit instantaneously. Attenuating delay provides fine-grained control over spikes as they propagate throughout a network. Our second example, an adding machine, relies heavily on delays to maintain consistent network dynamics.

Example 1 (A Simple Filter)**.** Tuning decay is useful in accounting for variation or error in temporally encoded information. The following example highlights this benefit. Suppose we have some input stream of binary data $\{x_t\}$, and we are interested in finding 1s either adjacent or separated by a single timestep. That is to say, we are looking for patterns 11 or 101. We suggest a two-neuron system to compute the following boolean function:

$$f(t) = \begin{cases} 1 & x_t = 1 \text{ and } (x_{t-1} = 1 \text{ or } x_{t-2} = 1), \\ 0 & \text{otherwise.} \end{cases}$$

1) Input neuron I_1 spikes at time t if and only if $x_t = 1$.

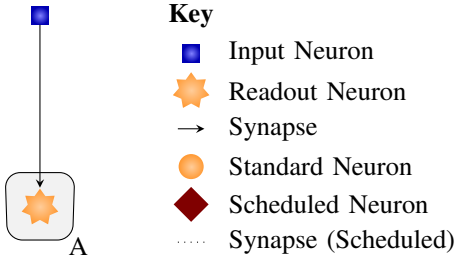

Fig. 3. The filter algorithm is illustrated. The key provided will be used for all such connection diagrams. Note that not all symbols are used in this diagram.

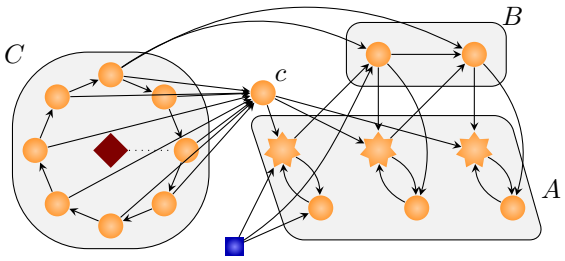

Fig. 4. A 3-register adding machine where all registers are size 8.

2) Readout neuron group A has one neuron defined by $(8.5, 8, .25)$, and there exists a single synapse $S_{I_0,a} = (1, 8)$.

The readout neuron A_1 spikes at time t if and only if $f(t-1) = 1$. The shift of one time unit is due to the synaptic delay. Table II lists an example run of the algorithm, and a connection diagram is provided in Fig. 3.

TABLE II
SAMPLE INPUT DATA AND THE RESULTING BEHAVIOR OF NEURON A_1

Value	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
I_1 spiked?	✓	✗	✓	✗	✗	✓	✓	✗	✗
V_{A_1}	0	8	2	8	2	.5	8.125	8	2
A_1 spiked?	✗	✗	✗	✓	✗	✗	✗	✓	✗

Example 2 (Approximating an α-adic Adding Machine)**.** This example uses spike-timing to efficiently represent an α-adic adding machine. The α-adic adding machine, sometimes referred to as an odometer or solenoid, is object of dynamical interest as it is equivalent, both from topological and ergodic viewpoints, to a wide variety of dynamical systems [31], [32]. From a more practical viewpoint, this allows for a fully spiking integer counter, which can prevent unnecessary transitioning between spiking and non-spiking regimes.

Since a true adding machine is infinite, we provide a k-register approximation. In addition, we assume each of the registers is of the same size, but generalizations from this assumption are clear.

Central to this algorithm is a spiking clock mechanism. Values are stored in the relative difference between spike time of a readout neuron and that of the '0-tick' on the clock. The clock can be provided by the regular spiking of a neuron determined by some external clock, or we can use a delay-spaced ring of neurons to obviate the need for an external clock. Differences in the times of spikes are used in biological neural nets, as well. For example, the well-studied spike timing dependent plasticity is a learning mechanism whereby influence is weighted relative to difference in spike times [33], [34].

Formally, for some positive integer α, we represent the space $\{0, \dots \alpha - 1\}^k$ as tuples $x = (x_0, x_1, \dots, x_{\alpha-1})$. Addition is modular, component-wise with carry-over. The standard adding machine map is $f(x) = x + (1, 0, \dots 0)$. To maintain brevity, we do not discuss the topology placed on this space, but we do caution that f does not iterate through elements according the ordering on the space. These differences must be taken into account depending on if implementing a dynamical adding machine or an integer counter.

The network is designed as follows:

1) A group C of α neurons forms our clock mechanism. Each neuron is assigned a value, in order. Each of these neurons have threshold 1, and the group is connected by
$$S_{C_i,C_{i+1} \mod \alpha} = (1, 1).$$
We, in addition, require another neuron connected to the 0-neuron in C which starts the clock and a neuron c which transmits the clock's timing.
2) A group of registers A stores the value of our adding machine. Each neuron represents one component of the tuple. The value is determined by the time the neuron spikes relative to the clock. We define, for $i = 1, 2, 3$, $A_i = (\alpha, 0, 1)$ and
$$S_{c,A_i} = (1, 1).$$
Since incoming spikes are integrated before a spike/no-spike decision is made, we need neurons that act as a buffer in the event that a register is increased at the same time step it spikes. For $i = 4, 5, 6$, we set $A_i = (1, 0, 0)$ with synapses
$$S_{I_1,A_4} = (2, 1/2),$$
$$S_{A_{i-3},A_i} = (1, 1/2),$$
$$S_{A_i,A_{i-3}} = (1, 1).$$
In addition, for $i = 1, 2$ we have synapses $S_{B_i,A_{i+4}} = (2, 1/2)$.
3) A single input neuron I_0 fires when iterating f. The neuron I_1 is connected to A_1 with weight 1 and B_1 with weight $1/3$.
4) For carryovers, we require $\alpha - 1$ neurons in a group which is denoted B each defined by $(1, 0, 0)$ with synapses
$$\forall j,\ S_{B_{j-1},B_j} = (2(j-1), 1/3),$$
$$\forall i, j,\ S_{A_i,B_j} = (1, 1/3),$$
$$\forall j,\ S_{C_{\alpha-1},B_j} = (2i, 1/3).$$

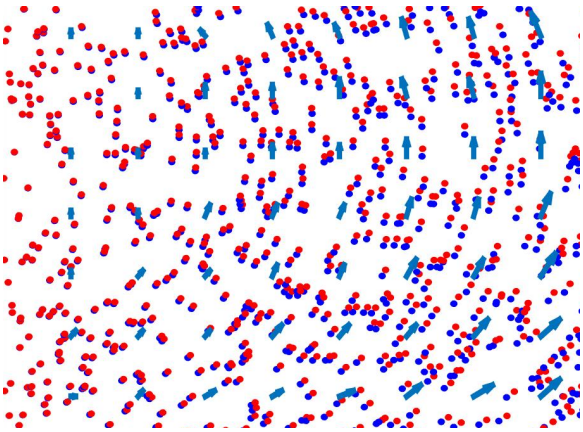

Fig. 5. A sample result of PIV computation. The blue and red dots represent particles in the first and second exposures respectively. Computed local flow vectors are shown superimposed.

The previous vignettes provide basic examples of spiking algorithms where spike timing, synaptic delay and decay complement the neurons' integration process to achieve computation. By combining these fundamental techniques with a highly parallel network structure, we can compute more sophisticated functions. In the following section, we discuss, in depth, spiking algorithms designed for a specific scientific application.

IV. Particle Image Velocimetry

Studying fluid flow is central to many fields in science and engineering. Particle image velocimetry (PIV) is a popular and well-studied method for using particles to determine the local velocity of a flow [35], [36], [37]. It finds use particularly in its ability to observe turbulent flow [35]. While there are many variations and adaptations, we look at the computations required for a standard one-viewpoint, two-exposure cross-correlation PIV method. Before observation, particles are injected into the flow. Two images of the same viewpoint are taken separated by a known time delay. The images are subdivided into smaller windows, each containing a handful of particles. The two frames are compared to determine an estimate for the local flow velocity. The method with which we are interested is cross-correlation. In general, the cross-correlation of two one-dimensional functions f and g is provided by

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(z)g(z+x)\,dz.$$

In the discrete case this reduces to

$$(f \star g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(m+n),$$

and the cross-correlation can be thought of as a sliding dot product. If we perform a cross-correlation on two (one-dimensional) images, the value $(f \star g)(n)$ gives the amount of overlap after a shift of n units. Following, the n for which $(f \star g)(n)$ is maximum gives the best estimate for the local velocity. We discuss the one-dimensional case for greater clarity. The algorithm is easily generalized to higher dimensions, and in practice we apply a two-dimensional version to the images of flow such as that shown in Fig. 5.

It should be noted that in traditional computing platforms the cross-correlation computation is accelerated by leveraging the Fourier transform. Specifically, by transforming into the frequency domain, the problem is greatly simplified.

A. An Initial Algorithm

Suppose f and g map $\{1, \ldots, k\}$ into $\{0, 1\}$. We design our spiking network as follows:

1) There are two input layers: F and G with k neurons each. At time-step 0, F_i fires if and only if $f(i) = 1$. Spikes in G are determined similarly with g.
2) We have a group of inner neurons: A with k^2 neurons arranged in a k^2 grid for convenience of notation. For all neurons in A, $T_{A_i} = 1$. We connect F_j to $A_{i,j}$ with a weight of .5. Furthermore, $G_{i+j-1 \bmod k}$ is connected to $A_{i,j}$ with a weight of .5:
$$\forall i, j,\ S_{F_i, A_{i,j}} = (1, .5),$$
$$\forall i, j,\ S_{G_{i+j-1 \bmod k}, A_{i,j}} = (1, .5).$$
3) An integrator group B of size $2k-1$ collects the spikes that occur within group A. The synapses are
$$\forall j \leq k, \forall i \leq k-(j-1),\ S_{A_{i,j}, B_i} = (1, 1),$$
$$\forall j \leq k, \forall i > k-(j-1),\ S_{A_{i,j}, B_{i+k}} = (1, 1).$$
The threshold of neurons in B must be set sufficiently high so that the neurons will not fire unless every pre-synaptic neuron fires, so we have $B_i = (2k, 0, 1)$.
4) Lastly, a single schedule neuron D that fires periodically connected to B and will, after some time steps, drive a neuron in B to fire. The neuron which has had it's voltage most greatly inflated by activity in A will be the first to fire. This neuron will correspond with maximum cross-correlation.
$$\forall i, S_{B_i, D} = (1, 1)$$

In examining the presented algorithm, a few points become clear. First, we use $O(k^2)$ neurons. Second, while the computation of the cross-correlation is constant time (twice the minimum synapse delay), the time required to compute the maximum is $O(k)$. We can reduce this to $O(k)$ neurons, $O(k)$ time by using time coding multiplexing.

B. Modifications for Constant-Time Performance

Note that while the overall algorithm runs in $O(k)$ time, the inner product is computed in constant-time. By switching to a constant-time maximum method, we can find the maximum cross-correlation in constant-time. This is easily accomplished using pairwise comparison. For example, to find if group X has greater activity than group Y, connect both groups to a readout neuron with connections from X being excitatory and

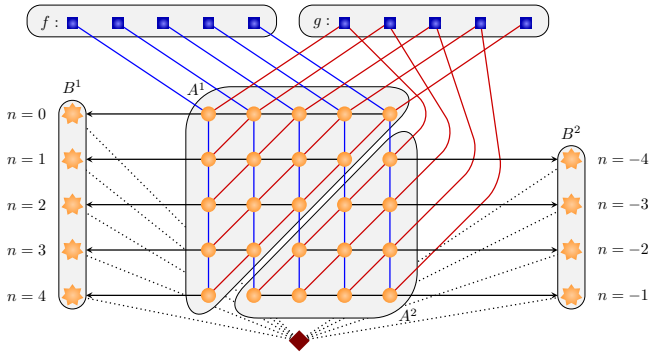

Fig. 6. A diagram of the first-try spiking neural network algorithm for cross-correlation. The first output neuron (star) to fire corresponds with the maximum. Neuron groups A and B are separated into sub-groups for clarity. The separations are $A = A^1 \cup A^2$ and $B = B^1 \cup B^2$.

those from Y being of equal weight but inhibitory. In the case of more than two groups for comparison, the group that wins all pairwise comparisons is the group with the maximum activity.

C. Algorithm Refinement Via Temporal Coding

By using graduated delay and time multiplexing, we can reduce the number of neurons required to $O(k)$ (precisely $4k+1$ neurons) while keeping the execution time at $O(k)$. The setup is slightly different, but the overall concept is the same. Essentially, each column (split by the diagonal) of the previous neuron group B is now represented as a time step.

1) There are two input neurons: F and G with 1 neurons each. At time-step t, F_1 fires if and only if $f(t) = 1$. Spikes in G are determined similarly with g.
2) Again, neuron group A computes the inner product with with $2k-1$ neurons defined by $A_i = (1, 0, 0)$. Neuron F_1 is connected to A_i with a weight of .5. All connections from F_1 have a delay of k time steps. Furthermore, G_1 is connected to A_i with a weight of .5 and delay i.

$$\forall i,\ S_{F_1,A_i} = (k, .5)$$

$$\forall i,\ S_{G_1,A_i} = (i, .5)$$

3) An integrator group B of size $2k-1$ collects the spikes that occur within group A. The threshold of neurons in B must be set sufficiently high so that the neurons will not fire unless every pre-synaptic neuron fires. Define $B = \{(2k, 1, 1)\}$ with synapses

$$\forall i,\ S_{A_i,B_i} = (1, 1).$$

4) We need a scheduled neuron D that fires periodically and connects to B. As before, the purpose is that after some time steps, at least one neuron in B is guaranteed to fire. The neuron which has had its voltage most greatly inflated by activity in A will be the first to fire. This neuron will correspond with maximum cross-correlation.

$$\forall i,\ S_{B_i,D} = (1, 1)$$

The description here computes a strict inequality, but simple weight adjustments can allow for non-strict inequality as well.

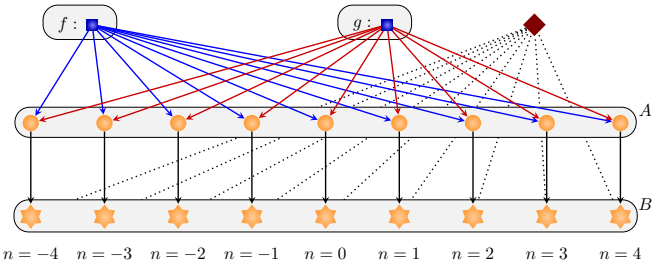

Fig. 7. A diagram of the temporal coding spiking network algorithm for cross-correlation. Due to multiplexing, only two neurons are needed for two inputs.

TABLE III
A SAMPLE EVOLUTION OF THE TEMPORAL CODING SPIKING NETWORK ALGORITHM

Neuron Group	Neuron Index	t_0	t_1	t_2	t_3	t_4	t_5	t_6
F	1	✓	✗	✓	✗	-	-	-
G	1	✗	✓	✗	✓	-	-	-
	1	✗	✗	✗	✓	✗	✗	✗
	2	✗	✗	✗	✗	✗	✗	✗
	3	✗	✗	✓	✗	✓	✗	✗
A	4	✗	✗	✗	✗	✗	✗	✗
	5	✗	✗	✗	✗	✗	✓	✗
	6	✗	✗	✗	✗	✗	✗	✗
	7	✗	✗	✗	✗	✗	✗	✗
	1	✗	✗	✗	✗	✗	✗	✓
	2	✗	✗	✗	✗	✗	✗	✗
	3	✗	✗	✗	✗	✗	✓	✗
B	4	✗	✗	✗	✗	✗	✗	✗
	5	✗	✗	✗	✗	✗	✗	✓
	6	✗	✗	✗	✗	✗	✗	✗
	7	✗	✗	✗	✗	✗	✗	✗
D	1	✓	✓	✓	✓	✓	✓	✓

Table III shows the propagation of spikes throughout the network based on sample input data. Since neuron B_3 is the first neuron in group B to spike, this corresponds to the maximum cross-correlation ($n = -1$).

In comparison to our first-try algorithm, this time coded algorithm is visually simpler, only requires $O(k)$ neurons, and runs in $O(k)$ time. Notice, however, that the number of spikes (which is a measure of the amount of energy) remains the same.

Subtle changes in decay rates allow this algorithm to have more generalized behavior. Specifically, the decay constant of the neurons in group A parameterizes the time scale over which they sample. Such tuning allows the algorithm to be more or less selective regarding variations in the timing of the input spikes.

D. Algorithm Analysis

The PIV algorithms exhibit trade-offs regarding space, time and energy costs, as shown in Table IV. The energy calculation is the expected number of spikes given each component of

each function is independent and is 1 with probability p. The variable κ_i is dependent on the input data and is defined as the number of shifts that correspond to the i-largest cross-correlation. The spikes for the input and schedulued neurons are omitted.

By using the time multiplexing and temporal coding in IV-C, we are able to effectively exchange neurons for timesteps, and the overall complexity is unchanged. Asymptotically both IV-A and IV-B require the same number of neurons, but IV-B computes in constant rather than linear time. Note however that accomplishing the maximum operation in constant time uses energy approximately quadratic in k.

TABLE IV
THE VARIOUS PERFORMANCE AND COST TRADE-OFFS OF THE DIFFERENT PIV ALGORITHMS ARE COMPARED

Algorithm	Number of Neurons	Runtime	Additional Energy Above $k^2p^2 + \kappa_1$
IV-A	$O(k^2)$	$O(k)$	0
IV-B	$O(k^2)$	$O(1)$	$\sum_{i=1}^{2k-1} \kappa_i(2k-1-i)$
IV-C	$O(k)$	$O(k)$	0

E. Generalizations

The first generalization that is of clear interest is the corresponding algorithm for a higher dimensional space. A two-dimensional version is straightforward; an implementation of the 2D algorithm using the Brian Simulator [38] produced the image in Fig. 5. Unfortunately, the connection diagrams and descriptions thereof become complicated and less enlightening. As such, they are omitted here. However, in experience gained from developing these systems, it has been easier to split the images along rows, with one input neuron per row. Though this is merely a convenience, and a system with no more than two input neurons is possible. Stereoscopic PIV exists, and this method generalizes nicely to three dimensions as well.

Robustness of the neurons used in the implementation can provide increased control. Particularly, different decay models can be used within neuron group A to control for possible errors in input timing or to increase flexibility in the flow detection.

Other generalizations lie in the individual component operations. The following are some examples:

- By setting g to be a reference pattern, rather than a second exposure, the algorithm performs a basic search for that pattern within a larger image.
- Replacing the neurons in group A with neurons that perform statistical measures, rather than an AND function, would allow for more holistic computations.
- After finding the cross-correlations, we find the maximum, but this operation can easily be replaced with other comparators such as a preference relation.

V. DISCUSSION

By presenting these "hand-crafted" algorithm designs, we hope to emphasize the computational benefits of a high-connectivity spiking network. Innate parallelization and colocalization of memory and processing are certainly critical for our algorithms' function, but in addition SNN-specific properties such as

- Spike timing,
- Synaptic Delay and
- Decay

have shown computational benefits of their own. We argue that these facets are central to the performance and energy benefits found with neuromorphic systems.

A second order component of the spiking approach is flexibility in data representation. Biological neural systems use a variety of methods to encode information; rate-coding, latency-coding, coding via inter-spike intervals and modular codes are all suggested theories [39], [40], [41], [42]. Computationally, the optimal representation is context dependent. In IV-C, we transition from a temporal code in the input data to what is essentially a rate-code in neuron group A (data is stored as the number of spikes over a given time) to a latency code in neuron group B. The flexibility of a spiking approach allows us to transition swiftly to the most appropriate representation at each step of the algorithm. The process works well in this context, but conversion, both in general and particularly from a SNN to von Neumann machine, is a costly operation.

We have chosen to develop these algorithms at an abstract, network level using a relatively generic neuron model. Thus, we target a wide range of options for eventual hardware implementation and acceleration, we receive the full benefit of a spiking model, and we avoid being tied a specific hardware platform. Some neuromorphic architectures do provide more sophisticated neuron models. Greater model complexity more accurately represents the true behavior of biological neurons, and the computational benefit of these models is an open question.

The next logical progression is the development of canonical neural computations—modular standardized operations that are well-suited for a neural approach. Thereby, one could implement high-level functions by combining these lower level components. Specialized high-performance, low-energy neuromorphic hardware would no longer be restricted to only machine learning and simulation but rather pose a viable optimization path for a wide variety of applications.

ACKNOWLEDGMENTS

This work was supported by Sandia National Laboratories' Laboratory Directed Research and Development (LDRD) Program under the Hardware Acceleration of Adaptive Neural Algorithms Grand Challenge. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

REFERENCES

[1] E. Knill, "Quantum computing with realistically noisy devices," *Nature*, vol. 434, no. 7029, pp. 39–44, 2005.

[2] C. H. Bennett and D. P. DiVincenzo, "Quantum information and computation," *Nature*, vol. 404, no. 6775, pp. 247–255, 2000.

[3] T. Pfeil, A. Grübl, S. Jeltsch, E. Mller, P. Müller, M. A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier, "Six networks on a universal neuromorphic computing substrate," *Frontiers in Neuroscience*, vol. 7, no. 11, 2013.

[4] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[5] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[6] J. Von Neumann and R. Kurzweil, *The computer and the brain*. Yale University Press, 2012.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[8] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3642–3649.

[9] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[10] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

[11] D. E. Feldman, "Synaptic mechanisms for plasticity in neocortex," *Annual review of neuroscience*, vol. 32, p. 33, 2009.

[12] J. B. Aimone, Y. Li, S. W. Lee, G. D. Clemenson, W. Deng, and F. H. Gage, "Regulation and function of adult neurogenesis: from genes to cognition," *Physiological reviews*, vol. 94, no. 4, pp. 991–1026, 2014.

[13] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Hfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. SAGHI, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen, "Neuromorphic silicon neuron circuits," *Frontiers in Neuroscience*, vol. 5, no. 73, 2011.

[14] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[15] K.-Y. Siu, V. Roychowdhury, and T. Kailath, *Discrete Neural Computation: A Theoretical Foundation*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[16] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, Aug 2013.

[17] E. Stromatias, F. Galluppi, C. Patterson, and S. Furber, "Power analysis of large-scale, real-time neural networks on spinnaker," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013, pp. 1–8.

[18] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 431–438.

[19] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[20] J. Park, S. Ha, T. Yu, E. Neftci, and G. Cauwenberghs, "A 65k-neuron 73-mevents/s 22-pj/event asynchronous micro-pipelined integrate-and-fire array transceiver," in *Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE*. IEEE, 2014, pp. 675–678.

[21] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *science*, vol. 338, no. 6111, pp. 1202–1205, 2012.

[22] M. Avery, J. L. Krichmar, and N. Dutt, "Spiking neuron model of basal forebrain enhancement of visual attention," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE, 2012, pp. 1–8.

[23] E. M. Izhikevich and G. M. Edelman, "Large-scale model of mammalian thalamocortical systems," *Proceedings of the national academy of sciences*, vol. 105, no. 9, pp. 3593–3598, 2008.

[24] J. Hasler and H. B. Marr, "Finding a roadmap to achieve large neuromorphic hardware systems," *Frontiers in Neuroscience*, vol. 7, no. 118, 2013.

[25] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie, "Silicon auditory processors as computer peripherals," *Neural Networks, IEEE Transactions on*, vol. 4, no. 3, pp. 523–528, 1993.

[26] M. Mahowald, *An analog VLSI system for stereoscopic vision*. Springer Science & Business Media, 1994.

[27] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[28] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1117–1125. [Online]. Available: http://papers.nips.cc/paper/5862-backpropagation-for-energy-efficient-neuromorphic-computing.pdf

[29] N. Joye, A. Schmid, Y. Leblebici, T. Asai, and Y. Amemiya, "Fault-tolerant logic gates using neuromorphic cmos circuits," in *Research in Microelectronics and Electronics Conference, 2007. PRIME 2007. Ph. D.* IEEE, 2007, pp. 249–252.

[30] H. Paugam-Moisy and S. Bohte, "Computing with spiking neuron networks," in *Handbook of natural computing*. Springer, 2012, pp. 335–376.

[31] L. Block and J. Keesling, "A characterization of adding machine maps," *Topology and its Applications*, vol. 140, no. 2, pp. 151–161, 2004.

[32] J. Buescu and I. Stewart, "Liapunov stability and adding machines," *Ergodic Theory and Dynamical Systems*, vol. 15, pp. 271–290, 4 1995.

[33] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *The Journal of neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[34] N. Caporale and Y. Dan, "Spike timing-dependent plasticity: a hebbian learning rule," *Annu. Rev. Neurosci.*, vol. 31, pp. 25–46, 2008.

[35] R. J. Adrian, "Twenty years of particle image velocimetry," *Experiments in fluids*, vol. 39, no. 2, pp. 159–169, 2005.

[36] A. K. Prasad, "Particle image velocimetry," *CURRENT SCIENCE-BANGALORE-*, vol. 79, no. 1, pp. 51–60, 2000.

[37] C. E. Willert and M. Gharib, "Digital particle image velocimetry," *Experiments in fluids*, vol. 10, no. 4, pp. 181–193, 1991.

[38] D. F. M. Goodman and R. Brette, "The brian simulator," *Frontiers in Neuroscience*, vol. 3, no. 26, 2009. [Online]. Available: http://www.frontiersin.org/neuroscience/10.3389/neuro.01.026.2009/abstract

[39] A. Thiel, M. Greschner, C. W. Eurich, J. Ammermüller, and J. Kretzberg, "Contribution of individual retinal ganglion cell responses to velocity and acceleration encoding," *Journal of neurophysiology*, vol. 98, no. 4, pp. 2285–2296, 2007.

[40] V. Kretschmer, F. Kretschmer, M. T. Ahlers, and J. Ammermüller, "High speed coding for velocity by archerfish retinal ganglion cells," *BMC neuroscience*, vol. 13, no. 1, p. 69, 2012.

[41] I. R. Fiete, Y. Burak, and T. Brookings, "What grid cells convey about rat location," *The Journal of Neuroscience*, vol. 28, no. 27, pp. 6858–6871, 2008.

[42] R. V. Rullen and S. J. Thorpe, "Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex," *Neural computation*, vol. 13, no. 6, pp. 1255–1283, 2001.